

LA-6943-H

History

C.3

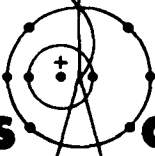
CIC-14 REPORT COLLECTION
**REPRODUCTION
COPY**

UC-32

Issued: May 1978

Computing at LASL in the 1940s and 1950s

Roger B. Lazarus
Edward A. Voorhees
Mark B. Wells
W. Jack Worlton



los alamos
scientific laboratory
of the University of California



LOS ALAMOS, NEW MEXICO 87545

An Affirmative Action/Equal Opportunity Employer

UNITED STATES
DEPARTMENT OF ENERGY
CONTRACT W-7405-ENG. 36

Printed in the United States of America. Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

Microfiche \$ 3.00

001-025	4.00	126-150	7.25	251-275	10.75	376-400	13.00	501-525	15.25
026-050	4.50	151-175	8.00	276-300	11.00	401-425	13.25	526-550	15.50
051-075	5.25	176-200	9.00	301-325	11.75	426-450	14.00	551-575	16.25
076-100	6.00	201-225	9.25	326-350	12.00	451-475	14.50	576-600	16.50
101-125	6.50	226-250	9.50	351-375	12.50	476-500	15.00	601-up	--1

1. Add \$2.50 for each additional 100-page increment from 601 pages up.

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

PREFACE

Each year the National Computer Conference devotes a session, known as Pioneer Day, to recognizing a pioneer contributor to the computing profession. The 1977 National Computer Conference was held in Dallas, Texas on June 13-16, 1977, and the hosts of the conference honored the Los Alamos Scientific Laboratory at the associated Pioneer Day. Since digital computation at LASL has always been a multifaceted and rapidly changing activity, the record of its history is somewhat fragmentary. Thus the 1977 Pioneer Day gave us the opportunity to survey and record the first 20 years of digital computation at LASL. Four talks were presented:

- I. "Hardware" by W. Jack Worlton,
- II. "Software and Operations" by Edward A. Voorhees,
- III. "MANIAC" by Mark B. Wells, and
- IV. "Contributions to Mathematics" by Roger B. Lazarus.

The contents of this report were developed from those talks. Each of them surveys its subject for the 1940s and 1950s. Together, they reveal a continuous advance of computing technology from desk calculators to modern electronic computers. They also reveal the correlations between various phases of digital computation, for example between punched-card equipment and fixed-point electronic computers.

During this period, LASL personnel made at least two outstanding contributions to digital computation. First was the construction of the MANIAC I computer under the direction of Nicholas Metropolis. The MANIAC system, that is hardware and software, accounted for numerous innovative contributions. The system attracted a user community of distinguished scientists who still enthusiastically describe its capabilities. The second development was an overt policy of the Atomic Energy Commission to encourage commercial production of digital computers. Bengt Carlson of LASL played a key role in carrying out this policy, which required close collaboration between LASL staff and vendor personnel, in both hardware and software development. Again, as you read these four papers, you see the beginnings of the present multibillion-dollar computer industry.

The Computer Sciences and Services Division of LASL thanks the National Computer Conference for recognizing LASL as a Pioneer contributor to the computing profession. The men and women who were at Los Alamos during the 1940s and 1950s are proud of this recognition, and those of us who have subsequently joined the Laboratory see in it a high level of excellence to be maintained. We also thank J. B. Harvill, of Southern Methodist University, for his collaboration and assistance with arranging the 1977 Pioneer day.

B. L. Buzbee



ABBREVIATIONS AND DEFINITIONS

ADP	automatic data processing
ALGAE	a LASL-developed control language for programming
ASC	Advanced Scientific Computer (TI)
CDC	Control Data Corporation
COLASL	a LASL-developed programming language and compiler (STRETCH) based on ALGAE and the use of natural algebraic notation
CPC	Card-Programmed Calculator (IBM)
CPU	central processing unit
Dual	a LASL-developed floating-point compiler for the IBM 701
EDSAC	electronic discrete sequential automatic computer
EDVAC	electronic discrete variable automatic computer
ENIAC	electronic numerical integrator and calculator
ERDA	Energy Research and Development Administration
FLOCO	a LASL-developed load-and-go compiler for the IBM 704
IAS	Institute for Advanced Study
IBM	International Business Machines Corporation
I/O	input/output
IVY	a LASL-developed load-and-go compiler for the IBM 7090 and IBM 7030
JOHNNIAC	John's (von Neumann) integrator and automatic computer
LASL	Los Alamos Scientific Laboratory
Madcap	a LASL-developed natural language compiler for the MANIAC
MANIAC	mathematical and numerical integrator and computer

MCP	Master Control Program, a LASL-IBM designed operating system for the IBM 7030
MQ	Multiplier-Quotient, a register used in performing multiplications and divisions
OCR	optical character recognition
PCAM	punched-card accounting machine
SAP	SHARE Assembly Program for the IBM 704
SEAC	Standards eastern automatic computer
SHACO	a LASL-developed floating-point interpreter for the IBM 701
SHARE	an IBM-sponsored users group
SLAM	a LASL-developed operating system for the IBM 704
SSEC	Selective Sequence Electronic Calculator
STRAP	STRETCH Assembly Program
STRETCH	IBM 7030, jointly designed by IBM and LASL
TI	Texas Instruments Company
UNIVAC	trademark of Sperry Rand Corporation

COMPUTING AT LASL IN THE 1940s AND 1950s

by

Roger B. Lazarus, Edward A. Voorhees,
Mark B. Wells, and W. Jack Worlton

ABSTRACT

This report was developed from four talks presented at the Pioneer Day session of the 1977 National Computer Conference. These talks surveyed the development of electronic computing at the Los Alamos Scientific Laboratory during the 1940s and 1950s.

I HARDWARE

by

W. Jack Worlton

A. INTRODUCTION

1. Los Alamos: Physical Site

Project Y of the Manhattan Engineer District was established at Los Alamos, New Mexico, in 1943 to design and build the first atomic bomb. Los Alamos occupies the eastern slopes of a volcanic "caldera," that is, the collapsed crater of an extinct volcano that was active some 1 to 10 million years ago. During its active period the volcano emitted about 50 cubic miles of volcanic ash, which has since hardened and been eroded to form the canyons and mesas on which the Laboratory is built.

Before its use for Project Y, this site was used for a boys' school, and those buildings were part of the

early residential and laboratory facilities. Old "Tech Area 1," where the early computers were housed, was built next to the pond, as shown in Fig. I-1. The early IBM accounting machines were housed in E Building, and the MANIAC was built in P Building. Since that time the computing center has been moved to South Mesa.

2. Computers in 1943

In 1943 computer technology was in an extremely primitive state compared to the rapidly growing needs of nuclear research and development. The analog computers in use included the slide rule and the differential analyzer. The slide rule was the ubiquitous personal calculating device, and LASL



Fig. I-1.

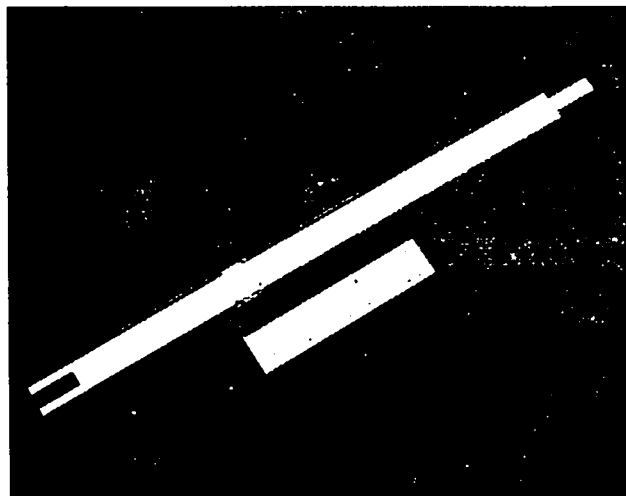


Fig. I-2.

made 18-in. slide rules available from stock. The example shown in Fig. I-2 belongs to J. Carson Mark, formerly head of the Theoretical Division at LASL. Although slide rules have now been largely replaced by electronic calculators, they once played an important role in computational physics—one which is still important and often overlooked. Mark points out that computers can be downright dangerous if their results are not checked with preliminary estimation techniques. In other words, estimation should precede computation. Using the results of computation without at least a rough estimate of

what the answer should be will inevitably lead to technical errors, because flaws in the models, the codes, or the data are essentially impossible to eliminate in the very complex models used at the frontiers of scientific research and development. In those early years when computers were less trusted than they are now, "Carson's Caution" was well understood, but we now accept computers so readily that we sometimes forget this very basic lesson from the past.

The other analog computing device used in the early 1940s was the mechanical "differential analyzer." These were one-of-a-kind devices not readily available, and thus they were not used at LASL.

Digital computing in those early days was done either with electromechanical desk calculators or with accounting machines. Both of these methods were used in the early weapons calculations.

3. Chronological Overview

Figure I-3. shows the various categories of computing devices that have been used by LASL from

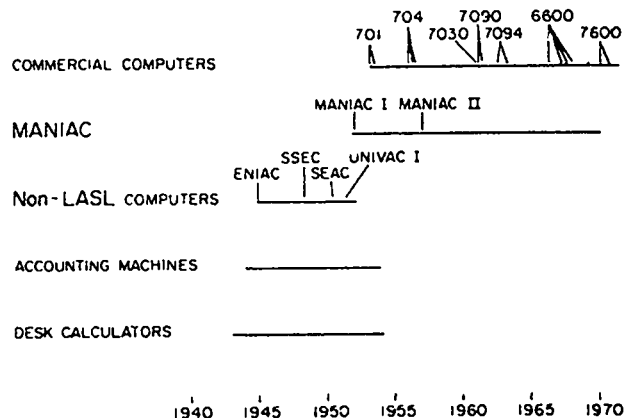


Fig. I-3.

the 1940s to the early 1970s. Note that not all of these have been used at LASL; in the late 1940s and early 1950s, some of the unique early computers at other sites were used in an attempt to complete some of the more critical calculations. In a sense, the "modern" era of LASL computing began in 1953 with the installation of the first IBM 701; this installation ushered in a period in which the major

computing requirements at LASL (and other large computing sites) would be met with equipment developed by commercial vendors rather than with computers developed as one-of-a-kind devices by Government laboratories and universities.

B. DESK CALCULATORS

Shortly after scientists began arriving at Los Alamos in March 1943, a desk-calculator group was formed under the direction of Donald Flanders. This group (T-5) consisted of both civilian and military personnel, including WACs (Women's Army Corp), SEDs (Special Engineering Detachment), and the wives of scientists. By 1944 it was the largest group in the Theoretical Division, with 25 people. The calculators used were Marchants, Monroes, and Fridens, although Flanders soon decided that it would be best to have a standard calculator and selected the Marchant (however, two advocates of Monroes refused to give them up). Repair of the calculators was a continual problem at the isolated Los Alamos site, so many of the problems with a sticking calculator were solved simply by dropping the end of the offending device, in the military tradition of the "drop test." Jo Powers (T-1) also notes that when their problems became acute, they called Dick Feynman (later a Nobel laureate) who, according to Jo, could fix anything. Feynman has recently given a lecture* that recounts some of his experiences with early LASL computing.

To avoid problems with manual errors, many of the calculations were executed by more than one person, with intermediate check points to assure that no errors had been introduced. Flanders designed special forms to aid in the setup and execution of the calculations. These calculations were typically done by a manual form of parallel processing; that is, the problem would be broken down into sections that could be executed independently. It seems that parallel processing is part of the "roots"

*"Los Alamos from Below: Reminiscences of 1943-1945," adapted from a talk at Santa Barbara Lectures on Science and Society, 1975. Published by *Engineering and Science*, January-February 1976, pp. 11-30.

of scientific computing, rather than just a recent innovation, as sometimes thought.*

C. PUNCHED-CARD ACCOUNTING MACHINES

The PCAMs of the early 1940s were designed primarily for business applications, but they could also be used for scientific calculations, such as the pioneering work of Comrie.** In early 1944, Stan Frankel (who worked with Metropolis) recognized that PCAM equipment could be used for some of the calculations at LASL, and that spring the following equipment was delivered:

- three IBM 601 multipliers
- one IBM 405 alphabetic accounting machine
- one IBM 031 alphabetic duplicating punch
- one IBM 513 reproducing punch
- one IBM 075 sorter
- one IBM 077 collator.

The 601 multiplier was the "workhorse" of this array of equipment. Its basic function was to read two numbers from a card, multiply them together, and punch the result on the same card, although it could also add and subtract (division was done by multiplying with reciprocals). The 601 was an important advance over its predecessor, the IBM 600, because the 601 had a changeable plugboard that made changing the functions being performed very rapid compared to rewiring the 600 for every such change. The 405 could add or subtract and list results. The 031 was a "keypunch" in modern terminology.

Early accounts of computations with these machines indicated that a single problem took about 3 months to complete; later methods reduced the time so that nine problems could be completed in a 3-month period.

*The ENIAC, the first electronic computer, employed parallel execution in its design.

**L. J. Comrie, "The Application of Commercial Calculating Machinery to Scientific Computing," in *Math Tables and Other Aids to Computation*, Vol. II, No. 16, October 1946, pp. 149-159.

The first of a series of the new IBM 602 Calculating Punch models was delivered to P. Hammer (Group Leader, T-5) on November 6, 1947, along with a new model of the sorter, the IBM 080. Although still an electromechanical device, the 602 had over 100 decimal digits of internal storage and could perform the following operations; add, subtract, multiply, divide, compare, and negative test. Property records indicate that eight 602s were delivered to Los Alamos in 1947 and 1948, and both 601s and 602s were in use for some time.

IBM's first electronic calculating punch was the 604, which was also used at LASL. It had 50 decimal digits of internal vacuum-tube register storage, although the sequencing control was through a prewired plugboard. Input and output was through punched cards, read and punched at a rate of 100 cards per minute. The 604 could perform the same operations as the 602, plus zero test, positive or negative test, repeat, and shift.

The IBM CPCs were delivered to Los Alamos in 1949; eventually LASL had six of these, the last of which was removed in October 1956. The CPC employed both electronic and electromechanical technology, with 1400 vacuum tubes and 2000 relays. It had 290 decimal digits of internal vacuum-tube registers plus up to three IBM 941 storage units that provided sixteen 10-digit words of relay storage each. The card reader and printer operated at 150 cards per minute and 150 lines per minute, respectively. The operations performed by the CPC included add, subtract, multiply, divide, repeat, zero test, suppress, shift, plus wired subroutines for transcendental functions

D. NON-LASL COMPUTERS USED FOR LASL STUDIES

From 1945 until the completion of the MANIAC, several non-LASL machines were used for LASL weapons studies. Nick Metropolis and Stan Frankel used the ENIAC at the Moore School of the University of Pennsylvania (before its being moved to Aberdeen, Maryland) for the first significant thermonuclear calculation. This was useful not only to LASL but also to the ENIAC project, because it gave this machine a rather thorough checkout. This

study was arranged by John von Neumann, who was a consultant to both the ENIAC project and LASL. Metropolis and Frankel collaborated on a study of the liquid-drop model of fission that used the ENIAC (also at the Moore School) in 1946 and 1947. In the summer of 1948, Foster and Cerda Evans (T-3), J. Calkin (T-1), and H. Mayer (Group Leader, T-6) used the ENIAC, this time at Aberdeen. As late as 1951-1952 Paul Stein also used the ENIAC for a LASL study.

The SSEC, a mixed-technology machine completed by IBM in 1948 in New York City, was used for LASL calculations by R. Richtmyer and L. Baumhoff in late 1948.

In 1951 the SEAC at the National Bureau of Standards in Washington, D.C. was used by R. Richtmyer, R. Lazarus, L. Baumhoff, A. Carson, and P. Stein of LASL.

The UNIVAC I at New York University was used by R. Richtmyer, R. Lazarus, and S. Parter. Paul Stein later used a UNIVAC I at Philadelphia.

Finally, the IAS computer was used by Foster Evans of LASL in a collaboration with von Neumann. Although the MANIAC was available at LASL by then, the work was completed on the IAS machine because the code was not portable to the MANIAC.

E. MAJOR COMMERCIAL COMPUTERS: 1953 TO 1977

In 1953 IBM delivered the first model of their 701 computer to Los Alamos, as arranged by John von Neumann who was a consultant to both organizations. Thus began the era at LASL in which computing requirements would be met with commercial computers. Figure I-4 shows the electronic computers that have been used at LASL from 1953 to the present. These computers have been (with the exception of the small CDC Cybers) the "supercomputers" of their time; that is, they were the most powerful computers available, where "powerful" is defined in terms of the speed of instruction execution and the capacity of the main memory.

Figure I-5 illustrates the trend in execution bandwidth in operations per second that has occurred at LASL from the early 1940s to the present.

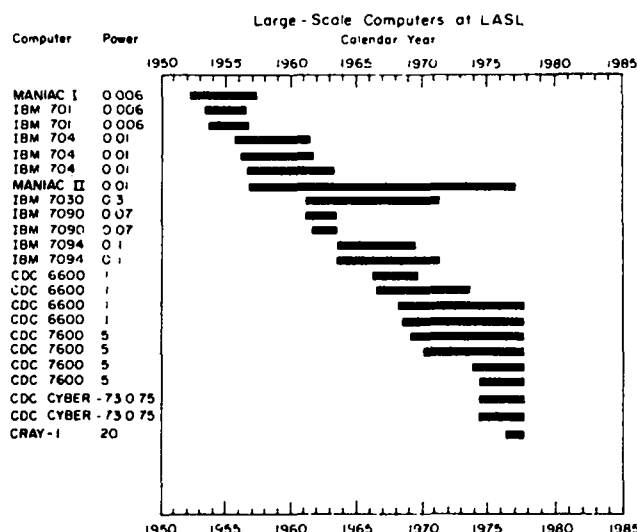


Fig. I-4.

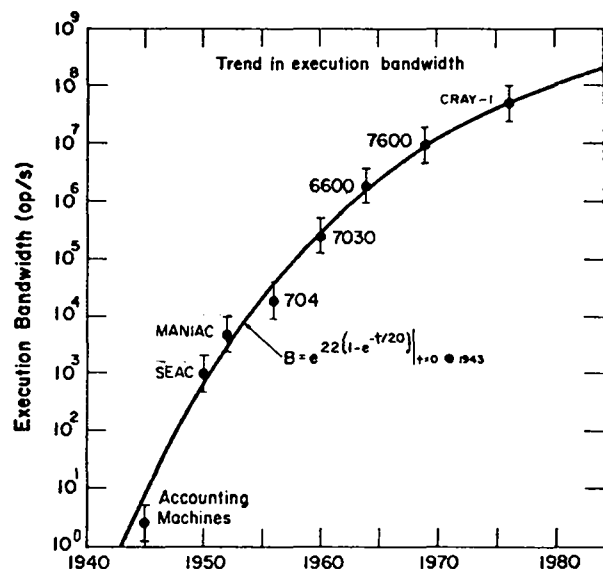


Fig. I-5.

The largest single step in this trend occurred with the development of the MANIAC, because this was a change from electromechanical to electronic technology. Whereas implosion calculations previously had required several months to complete on the PCAM equipment, the MANIAC was able to complete these calculations in a few days. Currently, CDC 7600s can complete this same type of calculation in a few hours. Overall, the trend covers

some 8 to 9 orders of magnitude in the change of execution bandwidth—a rare and perhaps unique change in technology in such a short period.

For many years LASL's growing needs for computational capacity and capability were such that an attempt was made to double the computing capacity every 2 years. This was successfully done from 1952 to about 1970, but the recent growth rate has not matched the former pace for two reasons: (1) the rate of development of faster computers is now somewhat slower than it was earlier and (2) the competition for computer funding within ERDA is much greater now than it was. LASL now competes with 38 other organizations within ERDA for ADP capital equipment funds.

The installation of ever-increasing computing capacity to match the growing needs of the research and development work at LASL would have been impossible if it were not for another important trend: the steady decline in the price-performance index of computers. The ENIAC cost about \$750 000 to build, and it executed about 500 operations per second. The most recent supercomputer at LASL, the CRAY-1, costs about 10 times as much as the ENIAC, but generates results about 100 000 times as fast. Thus, there has been a decline of some 4 orders of magnitude in the cost of an executed instruction in the 30 years since LASL began using the ENIAC at the Moore School.

The architecture of the commercial computers has changed in important ways in the 1940s and 1950s, including a trend toward ever-increasing levels of parallelism, as shown in Fig. I-6. The first

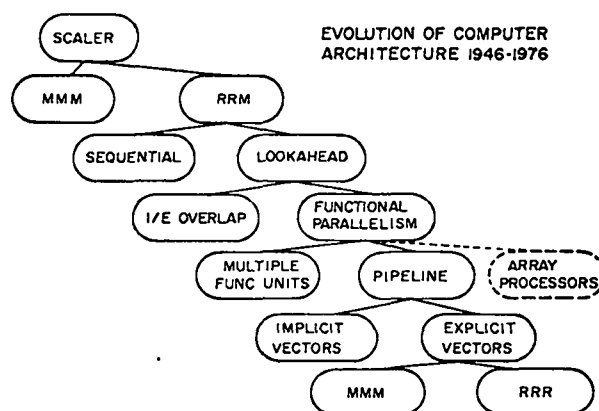


Fig. I-6.

computers following the ENIAC were strictly scalar processors; that is, they generated a single result for each instruction executed. Further, they were three-address designs, in which both operands used were addressed in memory and the result returned to memory (indicated by "MMM" in Fig. I-6). The so-called "von Neumann" type of architecture modified this addressing method by the use of an "accumulator" that allowed a single-address format to be used in which only one memory operand was addressed, and the other operand and the result were assumed to be in the accumulator (indicated by "RRM" in Fig. I-6).

Instructions were executed in strictly sequential mode in these early scalar processors: the instruction was fetched and decoded, the effective address was generated, the operand was fetched, the result was generated; the same sequence was then followed for the next instruction. With the design of the STRETCH computer (IBM 7030), which was a joint LASL-IBM effort, the instruction processing phase was overlapped with the execution phase through the use of a "lookahead" unit. Instruction processing then became less of a bottleneck, and the limit on operating speed became the "E-Box" that actually executed the operations (indicated by "I/E Overlap" in Fig. I-6). This limitation was rather quickly addressed in designs that included many "functional units," that is, independent units that could execute a designated function such as add or multiply independently of one another. The CDC 6600 had 10 such units, for example.

The next advance in architecture was the development of the "pipeline" processors, first acquired at LASL in the CDC 7600. This design breaks the instructions down into small segments. Each of these is executed independently and in parallel, thereby multiplying the execution rate by the degree of segmentation in the pipeline. Even this design is limited by the rate at which instructions can be issued, because only one result is generated per instruction. This bottleneck was addressed in the design of the "vector" processors, in which a single instruction can cause the generation of a set of results, using the same operation (for example, add)

in pairs on two *sets* of operands. The first-generation vector processors (the CDC STAR-100 and the TI ASC) were memory-to-memory designs in which operands were drawn from memory and the results returned to memory. This was, in effect, a repetition of the memory-to-memory design of the EDVAC, but in a vector context, and had the same disadvantage, namely that the memory and the central processor have widely disparate response times, thus making it difficult to avoid memory-limited performance. LASL carefully analyzed the first-generation vector processors and decided that the performance gain was too limited to justify the large amount of time and personnel effort involved in bringing one of these machines into productive status. A fourth 7600 was acquired instead.

The limitations of the memory-to-memory vector designs were addressed in the next generation of vector designs that used a register-to-register format, in which vector registers were included. These could be loaded from memory or have their contents stored in memory, but all vector operations drew operands from the high-speed registers and returned their results to the registers (indicated by "RRR" in Fig. I-6). LASL acquired the first of the CRAY-1 computers that included this design feature.

F. CONCLUSION

Nuclear science and computer science at LASL (and at other major research centers) have enjoyed a "symbiotic" relationship in which each has benefited from and contributed to the work of the other. Nuclear science has provided technical motivation and much of the funding for large-scale scientific computing, and computing has provided the tools and techniques for the solutions of many problems in nuclear science that would otherwise have been either intractable or much delayed. Computing remains a critical resource to the programmatic work at LASL, both for weapons research and development and for the growing efforts in energy research.

II SOFTWARE AND OPERATIONS

by

Edward A. Voorhees

A. INTRODUCTION

A more descriptive title would be "Software with Notes on Programming and Operations," because during the 1940s and 1950s, the coder (today called a Programmer or Systems Analyst) was usually the operator as well. Scientists normally programmed their own applications codes and on occasion might also code utility programs such as a card loading routine. They primarily used longhand, or machine language, in the larger codes to minimize running time. They were not afraid of "getting their hands dirty" and would do almost any related task to accomplish the primary work of math, physics, or any other field in which they were engaged. Some of this history was not well documented, and many of the old write-ups are not dated. Some details therefore could be slightly in error. I hope to convey a feeling for how computing was done in the "good old days" as well as to provide some information on the hardware and software available then.*

Hand computing was performed in the 1940s through the mid-1950s. At its peak there were perhaps 20 to 30 people using Marchant and Friden desk calculators. Mathematicians and physicists would partition the functions to be calculated into small calculational steps. These functions generally required many iterations and/or the varying of the parameter values. In many respects this constituted programming by the scientists for a "computer" that consisted of one or more humans doing the calculating, following a set of step-by-step "instructions."

The computing machines at LASL in the 1940s and 1950s fell into four groups. From 1949 to 1956, LASL used IBM CPCs; from 1953 through 1956, IBM 701s were also installed. These were all replaced almost overnight in 1956 by three 704s,

which remained until 1961. The IBM STRETCH computer arrived in 1961, but there was a period before that of about 5 years during which LASL did development work on both the hardware and software in cooperation with IBM.

B. CARD-PROGRAMMED CALCULATOR ERA

The IBM CPCs (used at LASL from 1949 to 1956) were not stored-programmed computers and were not originally designed as computers but rather as accounting machines. The card deck was the memory for the program and the constant data. The machine itself included 8 registers that held 10-digit decimal words. One could add up to 48 additional registers in units of 16 per box for a total of 3 such boxes. These were commonly referred to as "iceboxes." Figure II-1 shows an abbreviated version of the programming form developed for the CPC. The form could accommodate four different fields of operations; an operation normally consisted of Operand A, Operation, Operand B, and with the result being stored in the register identified in "C." Each of the wide gaps on the form indicate where two fields of data could be entered. The data were represented with a sign, a single integer, and seven fractional decimal digits. The exponent was represented as 50 plus or minus whatever the actual exponent would be. Branching was rather interesting. Even though each card was executed independently of every other, the machine could be

IBM CPC

(1)				(2)				(3)				(4)			
A	OP	B	C	A	OP	B	C	A	OP	B	C	A	OP	B	C
2	2	2	2												

Fig. II-1.

*I will discuss LASL's effort primarily, and references to IBM usually will be to indicate a joint effort or to maintain a frame of historical reference.

programmed to remember which of the four fields of operations it was following at any given time. Therefore, if the program had a branch (either unconditional or conditional) among the instructions in field 1, the machine could begin to execute its next instruction from any one of the other three fields. In this way you could branch between fields and follow a different sequence of instructions.

The design and use of the wired board "created" a general purpose floating-decimal computer from an accounting machine. The F Control Panels were LASL's most advanced wired boards and had many of the elements of "macrocoding."

Figure II-2 shows some of the operations that could be performed from a single card using the F Control Panels. Note that in these cases there are two operands and one result with a third operand, X, coming from another field of the form. Usually,

CPC FUNCTIONS

$$\sqrt{A}$$

$$A^n * B \ (1 \leq n \leq 10^8)$$

$$\sin X, \cos X, e^X$$

$$\frac{1}{2} \log \frac{1+X}{1-X}$$

$$\arctan X$$

$$\sinh X, \cosh X$$

Fig. II-3.

CPC SINGLE-CARD OPERATION EXAMPLES

$$\begin{aligned} A + B + X &\rightarrow C \\ \frac{B * X}{A} &\rightarrow C \\ (A + B) * X &\rightarrow C \\ \frac{X}{A * B} &\rightarrow C \end{aligned}$$

Fig. II-2.

these operations could be executed in one card cycle. Some of them would take longer than one cycle, so the programmer would have to put a blank card in the deck to give the machine enough time to execute that instruction before proceeding to the next one.

Figure II-3 shows some of the functions that were available on the CPC. The functions here and the operations in Fig. II-2 resemble modern-day subroutines, but they were subprograms wired on the board. Some of them were fairly complex, and often blank cards were again necessary to provide enough time for the CPC to execute the function.

Figure II-4 shows a board from an IBM 601. Although the CPC boards were much larger and more complex, this board shows some of the basic

features. Four wires come out of the columns identified as Brushes (which refer to the card-reading brushes) that go into the Multiplier. Another three



Fig. II-4.

wires go into the Multiplicand, and the Product is routed to the Card Punch. There were a total of six CPCs at LASL; the last one did not leave until late in 1956, well after the three 701s had already departed and three 704s had been installed.

The operator would stand in front of the machine, recycle the cards for hours, perhaps change to alternate decks of cards, watch the listing, and often "react." In other words, over 25 years ago we already had "interactive computing."

C. IBM 701 ERA

The IBM 701 originally was announced as the "Defense Calculator." LASL was the recipient of the serial number 1 machine, which arrived in April 1953 and remained until the fall of 1956. A second 701 calculator, as it was later called, came in February 1954. The machine was fixed binary. All numbers were assumed to be less than one; that is, the binary point was on the extreme left of the number. The electrostatic storage (which was not too reliable) could hold 2048 36-bit full words or 4096 16-bit half-words. It was possible later to double the size of that memory, which LASL did indeed do. Instructions were all half-words; data could also be in half-word format. Instructions were single address with no indexing. There was no parallel I/O. The system consisted of one card reader, one printer, one memory drum storage unit, two tape drives (which often did not work), and one card punch (the primary device for machine-readable output).

Figure II-5 shows the console of the 701. The Memory Register held word transfers to and from memory. If you wanted to add or subtract you would put one number into the Accumulator; if you wanted to divide or multiply, you would use the MQ register. There were two overflow bits in the Accumulator register in case of a spill, a condition the program could sense with an instruction. A 72-bit product was formed when multiplying two full words. A 72-bit dividend, in the Accumulator and MQ, divided by a 36-bit divisor yielded a 36-bit quotient in the MQ and a 36-bit remainder in the Accumulator. Six Sense Switches in the upper right-hand corner of the console could be interrogated by the program so as to alter the course of the program and thereby provided six manually set conditional branches. There were also four program-

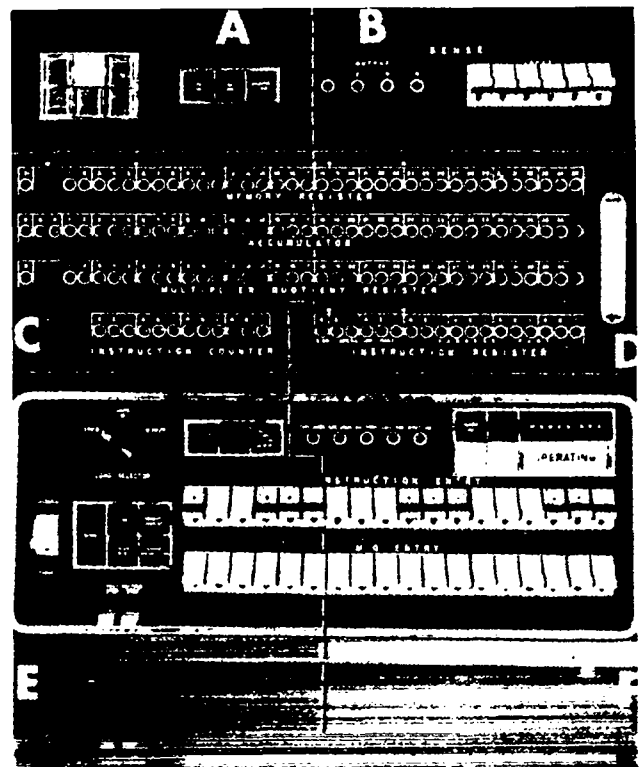


Fig. II-5.

mable Sense Lights, which could be used to visually indicate where in the code execution was occurring or some other internal condition. In the extreme lower left-hand corner, two buttons, one labeled Half Step and the other Multiple Step, permitted the programmer to step his way slowly through his program and observe the contents of the Accumulator, MQ, and the Memory Register when the 701 was switched to Manual Mode.

Before the first IBM 701 had been delivered, three principal methods of programming had been developed at LASL: longhand (which is perhaps better understood today if called "machine language"), SHACO, and Dual. SHACO and Dual were originated and implemented at LASL. All these programming systems were developed by 8 to 10 people and were operational soon after the delivery of the first machine.

Machine language came into operation in 1953 and was based on an early IBM assembly program called SO2. LASL's first version, "606," was soon followed by "607," which was then used during the remaining service of the 701s at LASL. Both used

"Regional Programming." Each location and address occupied a 3-column region field, which designated a block of instructions or data, followed by a 4-digit sequence number, which permitted up to 9999 instructions per block. A signed 2-digit

Fig. II-6.

single-address operation code was always expressed numerically. On the form (Fig. II-6), there were comment fields for a program label and the name of the operation. The absolute locations of blocks were specified by the coder for the assembly program at the time of loading for each of the different regions used. These assignments were on "origin cards." The card output options from the assembly program were absolute binary, regional binary (which was relocatable), and regional decimal (which was used to correct the source language deck and then punch a new regional decimal deck). The regional decimal punching option was too slow, so it was omitted from a later version of the assembly program. The output listing had both decimal and octal absolute locations; the octal were more useful than the decimal because most of the users worked in binary and octal. Scaling for fixed-point binary coding was generally noted in a comment field. The "607" assembly program was loaded at the beginning of the user's deck because there was no machine-resident software. Card errors were often fixed by plugging unwanted holes with chips from a card punch hopper. Some people got so adept at this that they could even fix the check-sum on binary cards.

Machine language allowed the user to get intimately close to the computer. There were no

monitors or other software stored in the computer. Everything that was in the computer was there because the user loaded it from cards. When he got on the machine, he loaded a bootstrap card loader that loaded the rest of his deck. He loaded his own print program (fixed output format) and manually put the corresponding print board in the printer. If there was trouble on a run, he then loaded an appropriate debugging program and associated printer board of his choice (generally, either a printed dump or an instruction-by-instruction tracing of a portion of the code as it executed). Memory check-sums and frequent dumps were made as protection against the short mean time (minutes) between computer failures.

With machine language, the user had to remember the following.

- When dividing, the divisor always had to be less than the dividend so that the quotient would be less than 1. If not, a Divide Check would occur and the machine would come to a screeching halt.

- An instruction was often used also as a data constant; that would be unheard of today.

- Because the 701 was a fixed-point binary machine, the user had to think in binary octal, especially when working at the console or poring over an octal dump.

- Scaling was necessary and often difficult to do without causing undue loss of significance or result overflows. The programmer had to mentally keep track of the binary point as he programmed.

- When decimal data were loaded, the programmer specified both the decimal and binary points for the conversion. For example, the integer 13 (decimal) would have a decimal scaling factor of 2 (for two decimal digits), but the binary scaling factor would have to be at least 4 to accommodate the number of binary bits from the conversion.

- When adding two numbers, they had to have the same binary scale factor for proper alignment; otherwise, the user would have to shift one of them until they were aligned. He also had to allow room for a possible carry or check the Overflow Indicator after the addition.

The Program Library included various card loaders, print programs, and debugging programs. The debugging programs would look for such things as transfers or stores to certain specified locations. Memory errors frequently resulted in a Divide

Check and a machine stop. Occasionally, instruction sequence control would be lost, and a jump would occur to some part of memory where it should not be. In this case, when the machine stopped you had no idea how control got to that location.

LASL Group T-1, which ran the computer operation, began issuing materials and offering programming classes in 1953. At that time there were about 80 users outside of T-1. By August, LASL was already operating 24 hours per day. Six 701s had been delivered nationwide by then, and there was enough interest among users to hold a program-exchange meeting at Douglass Aircraft (Santa Monica). This meeting was the forerunner of the SHARE organization formed in 1955 of which LASL was a charter member.

SHACO was an attempt to simulate the widely used CPC decimal coding scheme. It was an interpreter with an option for tandem arithmetic operation. SHACO was 20 to 60 times faster than the CPC depending on the amount of printing done during execution. Printing on the CPC was overlapped with execution and hence was "free." SHACO was 2 times slower than the 701 when executing machine-language codes that incorporated floating-point subroutines. It was about 10 to 15 times slower than a good 701 machine-language fixed-point code. If the tandem arithmetic option was used, execution was slowed by another factor of 2. SHACO's floating-decimal data representation was 10 digits (1 integer plus 9 fractional digits) and a 3-digit signed exponent (not modulo 50). It had a maximum of 24 instruction blocks each of which could contain up to 127 instructions. There was also a maximum of 705 data locations. Data exponents were stored separately from mantissas. Figure II-7 shows the input format. SHACO was the forerunner

of IBM's Speedcoding system, which was issued in 1954. Their version was very similar, and in their manual they acknowledged that LASL had completed a program with the same objective of Speedcoding: to minimize the amount of time spent in the problem preparation. SHACO, although considerably slower than 701 machine language, was very effective for short problems where few runs were anticipated and/or for "exploratory" difficult-to-scale codes. Using a language of this type on the 701 saved approximately a factor of 20 in coding and debugging time.

Dual was a LASL-developed fixed- and floating-decimal coding language that came out about the same time as SHACO. Its authors claimed that it would "transform the 701 into a combined floating-point and fixed-point computer." Its commands strongly resembled 701 assembly-language single-address commands. It had a "virtual" combination of the Accumulator and the MQ into a single AMQ universal register. It had built-in tracing and a single-address coding format. Dual executed commands by branching to subroutines that occupied about a quarter of the electrostatic storage. It represented its data with a modulo 50 exponent written in front of the mantissa. For example, the decimal number -3 would be written as -51.300. It had a limited set of functions, such as square root, cosine, and exponential. Both Dual and SHACO were extensively used for all programs other than the very large production codes.

There were no machine room operators as such; there was a dispatcher who kept records of usage, downtime, and the schedule of assigned users. A user operated the computer while his programs were running (Fig. II-8). During the day, he would request and was allocated 2- to 5-minute periods on the computer for debugging, checkout, etc. When his turn came, he would have his program card deck plus debugging programs in hand. The preceding user would normally be printing or punching at the end of his run, and he would usually allow the next user to stack his cards in the card reader. When the preceding user finished, the next user would be ready to initiate loading of his deck. Sometimes the preceding user would overrun his allocation. Usually, he would be allowed to continue for another

SHACO FOR 701

BLOCK	CARD NO.	±	A	OP	±	B	C
2	3	1	3	2	1	3	3

± X.XXXXXXXXXX ± EEE (EXPONENT STORED SEPARATELY)

Fig. II-7.



Fig. II-8.

minute but then the situation could get tense indeed. More than once, the waiting user pushed the CLEAR button to erase memory. The habitual overrunners developed a reputation for such; but, all in all, things went rather smoothly.

At night, users with long-running jobs were able to do something other than tend the computer. An AM radio was attached to the machine. Different sections of a code had a unique rhythm and sound so the user could do other work if he kept one ear half-cocked to the sound of the radio. If he heard something unusual, he took appropriate action. Night runs would range from 30 minutes to 4 or more hours. Occasionally, when oversubscribed, upper management might have to decide who would get the time available during the night.

D. IBM 704 ERA

The IBM 704, initially announced as the "701A" in mid-1954, came to LASL in January 1956. It represented a significant improvement over the IBM 701. For one thing, instead of electrostatic storage, it had core memory, which was much larger (up to 32k words) and far more reliable. It had floating-point binary, so SHACO and Dual were unnecessary. It had index registers; although there were only three, they could be used simultaneously in combination if you were a very tricky coder. It had a much larger vocabulary and some logical operations. It was 2-1/2 times faster. By September 1956, three 704s had been installed at LASL and two 701s released. There was a lot less procurement red tape at that time.

LASL-developed 704 software included longhand (assembly language), FLOCO, ALGAE, and SLAM. The 704 machine language, called Regional Symbolic or LASL "871," was a straightforward assembly program based on the previous assembly program, "607," for the 701. However, it had alphabetic operation codes and nonsymbolic numeric locations. It had additional fields for the new index registers (Tag). The Decrement was the field used to modify the contents of index registers. It was possible to partially assemble a deck and then merge it with the previously assembled binary deck. There was a greatly expanded subroutine library.

Eight months after the first 704 was installed, we issued the 704 Primer. Apparently we were in no hurry to enlist new computer users. The 701 users already had IBM manuals and needed little additional help. The 704 Primer, an extension of the early 701 Primer, was mainly used in the courses for beginning coders.

FLOCO, which used what we called a "flow code," came out in 1956 and was replaced by FLOCO2 in 1959. The idea was to create a single-address floating-decimal system that could be loaded and immediately begin to execute following a one-pass compilation that occurred during the card loading. This saved greatly on compile time. The slogan for FLOCO was that the source deck was the object deck. One could have up to eight instructions per card. The flow code controlled the execution of what was called the "formula set." FLOCO had pseudoinstructions interspersed in the card deck that caused alternation of loading and execution. The computer would load some instruction cards, execute the instructions, load more cards, and continue in this manner. Data or data space referred to by a formula had to precede the formula in the card deck to allow address assignment in the instructions during the single pass. For some reason that is not now clear to me, it was necessary to load data in backward order. One could have transfers, branches, or jumps within a formula but not directly to another formula. This had to be done indirectly by a return transfer to the flow code. Note that the flow code defined the flow of the program and was separate from the formula set or set of things to be done; that is, the logic of the overall code was not embedded in the rest of the code.

In 1958, the ALGAE language was implemented at LASL for the 704. It was a preprocessor to FORTRAN (which was first issued by IBM in 1957). The language contained a control structure that resembled arithmetic expressions. The basic idea was to separate the specification of the program control from that for the evaluation of equations, I/O statements, data movements, etc., reducing the common two-dimensional flow diagram to a compact linear statement (box labeled Control in Fig. II-9). That control statement plus the set of "things

ALGAE FORMULATION OF FLOW DIAGRAMS

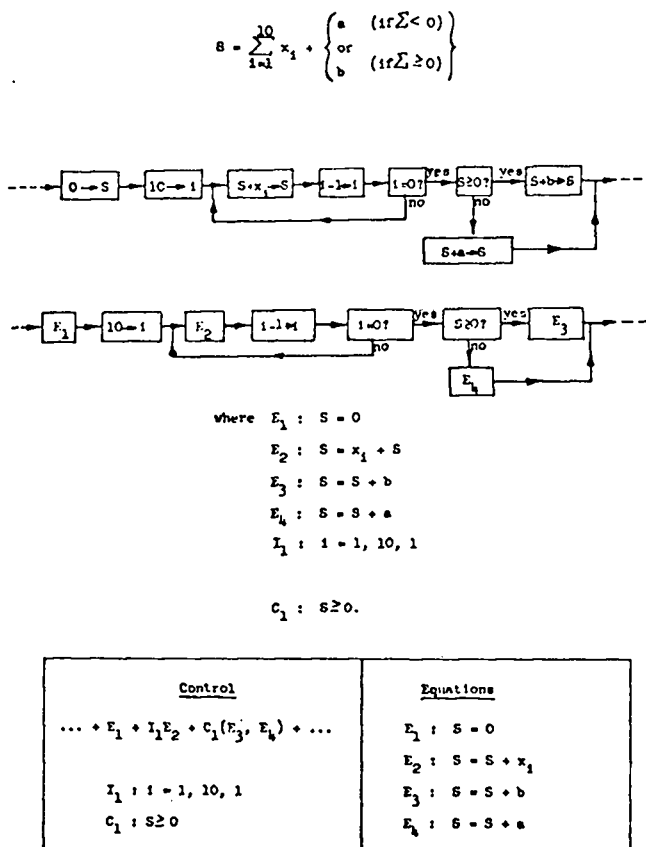


Fig. II-9.

to be done" totally represent and define everything that is in the flow diagram. A paper on the language published in Vol. 1 of *Communications of ACM* points out that the GO TO statement is probably unnecessary in nonmachine programming languages. Hence, 20 years ago, there was a suggestion of GO-TO-less programming in the literature.

SLAM was an elementary monitor program for the 704. It accepted multiple jobs batched on tape and produced batched output on tape for offline printing. It removed the necessity for a coder to be in attendance during his run. It accepted all the various languages in operation at LASL then. Its use was not mandatory. Larger codes would use single-purpose tape-in and tape-out utility programs for offline purposes instead of SLAM.

E. STRETCH ERA

Next came the planning period before the delivery of the STRETCH computer (IBM 7030). STRETCH was a major LASL-IBM joint effort that began in 1956. The hardware was novel and revolutionary compared to the IBM 704 hardware. The vocabulary was enormous. The machine was delivered in 1961.

LASL and IBM worked together in the development of STRETCH software, including the design of a longhand assembly program named STRAP and an operating system called MCP. STRAP-1 (implemented by LASL in 1958) was actually a crosscompiler that operated on the 704 and produced code for the 7030. A 7030 simulator (IBM) operated on the 704 and incorporated STRAP-1. STRAP-2 (implemented by IBM in 1960) accepted the same language but ran on STRETCH itself.

Figure II-10 shows the evolution in the design of three assembly program languages. The problem is to compute $T = Z(X + S3)$ with floating-point arguments. Note that the "871" coding form for the 704 had fixed fields, whereas STRAP had free-form statements for instructions. Also note the steady progression from numerics to symbols in the three generations of assembly language.

The MCP for the 7030 was designed by LASL and IBM and was written by IBM. It had parallel batching of I/O and disk operations, which was referred to as "spooling." It handled interrupts; there were many on the 7030. IBM 1401s came into use for offline I/O support; there were four of them during the lifetime of STRETCH, beginning in 1960. There was no multiprogramming then, but the overlap of I/O and calculation was possible. MCP was not used by most of the nighttime, long-running production codes because it would have precluded I/O functions

LOCATION	OPERATION	ADDRESS	DECREMENT	DATA	REMARKS
REGION SEQUENCE		REGION SEQUENCE	REGION SEQUENCE	FRACTION	
0005 00000	CA	800 0		X INTØ AC	
1	FA	801 0		ADD S3	
2	ST	1 0		STØRE SUM	
3	LQ	1 0		SUM TØ MQ	
4	FM	900 0		MULTIPLY BY	
0005 00050	ST	850 0		STØRE T	

LOCATION	OP	ADDRESS, TAG, DECREMENT
1 2	3 4	5 6 7 8 9 10 11 12
CALCT	CL A	X
	FAD	S3
	STØ	TEMP
	LQ	TEMP
	FMP	Z
	STØ	T

STATEMENT NAME	STATEMENT
1 2	3 4
CALCT	L, X
	+ S3
	"Z: ST, T^A^" SEE^NOTE^1
	"SEE^NOTE^2^ON^NEXT^PAGE"

Fig. II-10.

being closely phased into the code by the programmer.

Two programming languages, COLASL and IVY, were developed totally by LASL concurrently with the joint LASL-IBM effort.

Figure II-11 shows the keyboard for the IBM 9210, which was built by IBM to specifications developed at LASL from 1956 to 1960. The goal was to develop an input device with a large number of characters and other features to use with "natural languages." It incorporated a triple-case typewriter with three letters on each type slug. It could superscript and



Fig. II-11.

subscript any number of levels. A sample 9210 output from the typewriter and card punch is shown in Fig. II-12. Note that three cards are needed to represent the equation

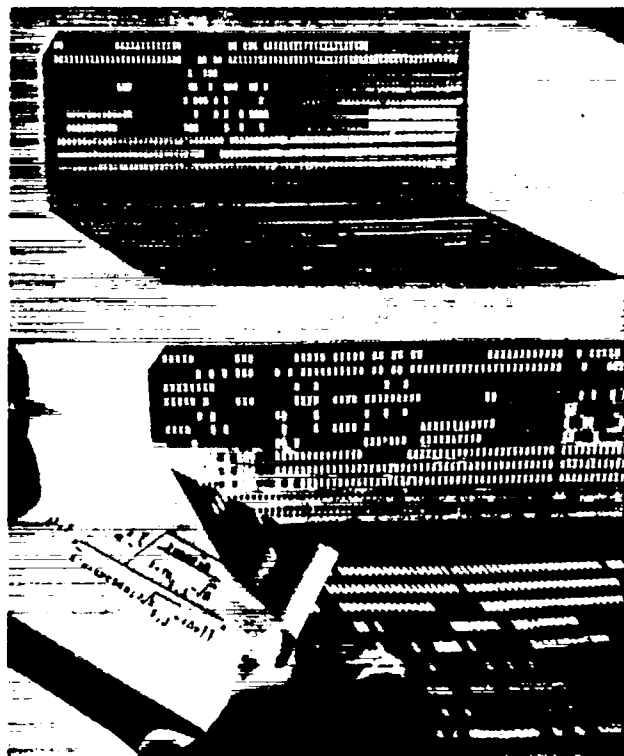


Fig. II-12.

COLASL (Compiler of LASL), based on the 9210, was developed and appeared in 1961. (The MANIAC had a similar compiler named Madcap with a more limited character set. In 1960 Madcap had superscripts and subscripts as well as displayed functions.) COLASL accepted "natural" mathematical notation. The code (typed in black) was often embedded in the narrative write-up or commentary, which had to be typed in red. Red characters were ignored during compilation. If desired, you could switch off the red key; everything would come out in black and would look like a report. COLASL was actually precompiled to FORTRAN. The COLASL statement analysis used some rather advanced techniques such as recursive subroutines, tree-structures, and a complete logic trace.

Figure II-13 shows an example of COLASL code as written by the programmer. An example of

STATEMENT NAME	STATEMENT
	Comments were written in red and were ignored by the compiler.
	COLASL could accept equations such as:
	$x = \sqrt{1-x^2} = \sqrt{1-x^2}$ or
	$P_i = (q_i^{1/2}, i = 1, 2, \dots, p)$

Fig. II-13.

THE SPHERICAL BESSEL ROUTINE

The routine, $J_p = \sqrt{x/ix} J_{p+1/2}(x)$ is represented by the notation: $J(x, \bar{J}, P)$, where x is a normalized floating point number, \bar{J} is the first word of the output array, $\bar{J}_1 \dots \bar{J}_P$, and the range of p is determined by the integer P , $P \geq 0$.

Definition:

Let $P = P + 10$, and $M = ix + 10$. If $(\bar{P} > M)$ then set $M = \bar{P}$, the larger of the two.

Since $J_M = 0$ as $M \rightarrow \infty$, the method consists of guessing

$J_M = 0$, $J_{M-1} = 10^{-7}$. The remaining J_i are then computed using the recursion formula

$$J_{i-1} = \frac{i+1}{x} J_i - J_{i+1}, \text{ for } (i = M-1, M-2, \dots, 1).$$

However, J_0 may be obtained directly from the relationship

$J_0 = \frac{\sin(x)}{x}$, and the normalization factor $R = \frac{J_0}{J_P}$ can be computed.

The remaining J_i are obtained by letting $\bar{J}_i = R J_i$, for $(i = 1, 2, \dots, P)$. This is the end of the routine definition.

Routine: $J(x, \bar{J}, P)$.

$P = P + 10$, $M = ix + 10$. If $(\bar{P} > M)$ then $M = \bar{P}$. $J_M = 0$, $J_{M-1} = 10^{-7}$.

$J_{i-1} = \frac{i+1}{x} J_i - J_{i+1}$, $(i = M-1, M-2, \dots, 1)$. $J_0 = \sin(x)/x$, $R = J_0/J_P$.

$\bar{J}_i = R J_i$, $(i = 1, 2, \dots, P)$. End routine.

The spherical Bessel routine in the COLASL language.

Fig. II-14.

COLASL source code made on the 9210 is shown in Fig. II-14. Note the use of \bar{J} and displayed quotients, which were not possible using FORTRAN. Neither of the LASL-built natural language compilers, COLASL or Madcap enjoyed any widespread success, in part because FORTRAN was already

well entrenched after 4 or 5 years. Another factor was the lack of an adequate and relatively inexpensive I/O device corresponding to the 9210. We could not afford to buy a second 9210 when we wished to do so later. Perhaps in the future, OCR or some other technology will begin to make the use of mathematical notation feasible.

IVY for the 7030 was based on a similar compiler by the same name for the IBM 7090. It came out in 1961 as a successor to the FLOCO language. Again it was a load-and-go one-pass compiler-assembler that attempted to combine machine language and an algebraic language based on Polish notation. It could relocate data while the code was executing, an option referred to as "dynamic storage allocation."

F. CONCLUSION

During the 1940s and 1950s programming emphasized machine efficiency. Codes were very machine-dependent. Ease of programming was a secondary consideration. Hands-on computer operation was the norm. Systems software, although primitive by today's standards, was generally ready for use when the computer was delivered (even when you were getting serial number 1 or serial number 2 hardware). Programs were freely exchanged. Some LASL-developed compilers were more popular elsewhere than they were at LASL. Programming was not ego-less; programmers took pride in their work. Competition to write smaller, faster math subroutines or utility programs was common. The users today are now insulated functionally and physically from the hardware. At LASL, special arrangements are now necessary even to see a big computer, and it is not clear to me that that is good. Today, conventions, standards, regulations, and procedures are far more abundant in procurement and in the use of computers. I believe there is good reason for concern about such restrictions because they can stifle progress in computer design, in software design, and in the use of computers. The industry is no longer driven by what the users think they need, but rather by what industry thinks they can sell.

Finally, it is interesting that there seems to be a revival of stand-alone or distributive computing, which I view to be a move to gain more control for the user. The progression has been from hands-on

computing with optimization of hardware utilization to batch (no hands-on) to timesharing (pseudo hands-on with no concern for hardware efficiency).

The trend now seems headed back toward increased interest in efficient hardware utilization.

III MANIAC

by

Mark B. Wells

I am going to reminisce a little about the MANIAC computer that served LASL so well in the early 1950s. Actually, there were three MANIAC computers. MANIAC I, which I will discuss mostly, was at LASL from 1952 to 1957. MANIAC II was there from 1957 to 1977. MANIAC III was never at LASL; it was built in the late 1950s at the University of Chicago where Nick Metropolis, the prime instigator for all three machines, spent a few years. (Figure III-1 shows MANIAC III under construction.) The word MANIAC is an acronym for

machine names, such as ENIAC and EDSAC, prevalent then. Now one wonders if it may have been a stimulus instead of a deterrent. The late George Gamow, well-known astronomer and physicist, had his own interpretation of the acronym. Talking to John von Neumann, he suggested that maybe MANIAC should stand for Metropolis and Neumann invent awful contraption.

The MANIAC I computer at LASL is often confused with the IAS computer at the Institute for Advanced Study in Princeton (Fig. III-2). At the plan-

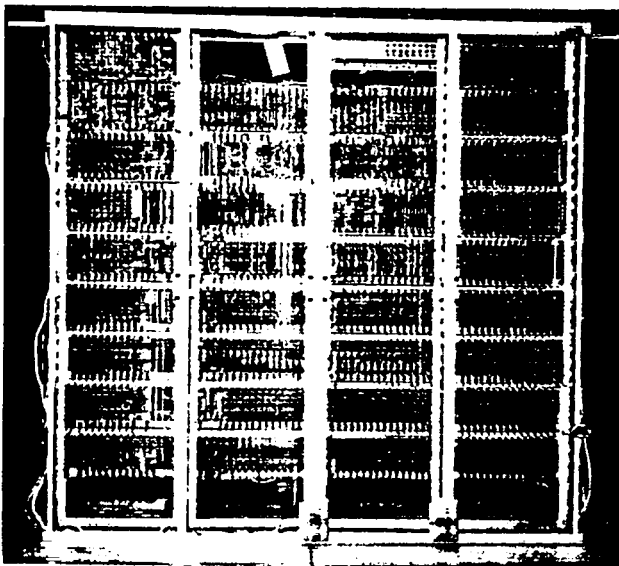


Fig. III-1.

mathematical analyzer, numerical integrator, and computer. Nick tells me that he chose the name partly in the hope of stopping the rash of acronyms for

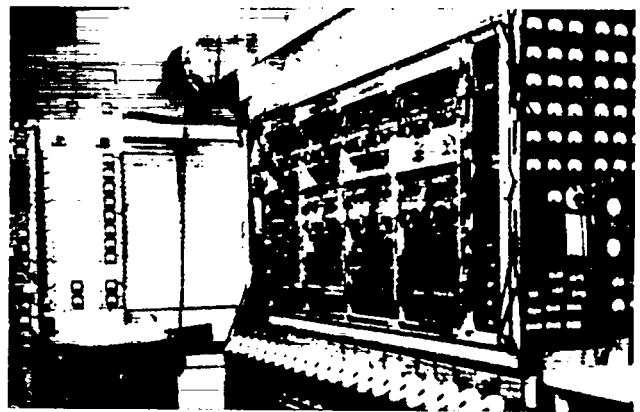


Fig. III-2.

ning stage in 1948, MANIAC was to resemble the machine being built in Princeton by von Neumann and Julian Bigelow. However, when it was completed in 1952, it was quite a different machine. One of the hardware differences was the size of the Williams tubes used for memory: 2-in. tubes on the MANIAC instead of the 5-in. tubes on the IAS computer. Two-in. tubes were chosen by Jim

Richardson, the chief engineer on the MANIAC project, because they required less space. Three-in. RCA tubes were substituted later. By the way, do not confuse either of these machines with the JOHNNIAC built at Sperry Rand Corporation by Willis Ware.

MANIAC I was a vacuum-tube machine powered by a room full of batteries (Fig. III-3). Figure III-4



Fig. III-3.

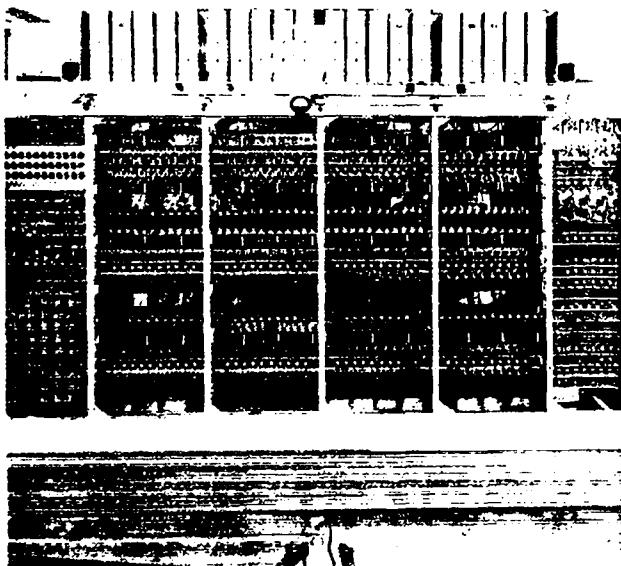


Fig. III-4.

shows MANIAC I. The arithmetic unit with the three registers is in the middle with the operation controls on the sides and in the back. The word length was 40 bits; if you look closely, you can see 10 bits of register in each of the 4 central panels. The memory is on top. There are 2 Williams tubes in each of the 20 boxes. The 2 monitor tubes at the ends were for viewing the contents of any of the 40 tubes. Each tube could store 32 by 32, or 1024, bits of information; hence, MANIAC I had a memory capacity of 1024 words. Out of the figure on the right, or perhaps it had not been installed yet, was a 10 000-word Engineering Research Associates drum for auxiliary storage. On the far left is the row of switches that served as the user's console as well as part of the engineer's console. The user's console was later moved to a table. The controls used by the engineers to tune the memory and view its contents were accessible on the front of the memory boxes and below the memory. I can still remember the Friday when those controls were recklessly twiddled by a brilliant, but rather naive, mathematician named John Holladay. After spending several hours that weekend readjusting the controls, an engineer (I believe it was the late Walter Orvedahl) installed some very attractive switches just below the memory. These "Holladay" switches were for the unauthorized person who could not resist twiddling; they did absolutely nothing.

Actually, there was good rapport between the engineers and the programmers, or coders as they were called in those days. Nick, who is still at LASL, and later Jack Jackson, now with IBM, were the primary design architects of MANIAC I and its system, but suggestions for hardware modifications as well as operational procedures were proposed arbitrarily by users or engineers. There are many examples of this user-engineer interaction throughout the service of MANIAC I and MANIAC II. One was Lois Cook Leurgans' naming of the hardware breakpoints "red" and "green" after the color of the pencils she used to indicate the temporary stopping points in her program. (She is shown at the console in Fig. III-5.) The suggestion for the successful console-controlled breakpoint, called the purple breakpoint, came from Bob Richtmyer, an avid MANIAC user, who is now with Colorado University. Also, it was interaction between engineers Jim Richardson and Grady McKinley and coders Bob Bivins and me that led to the development of the

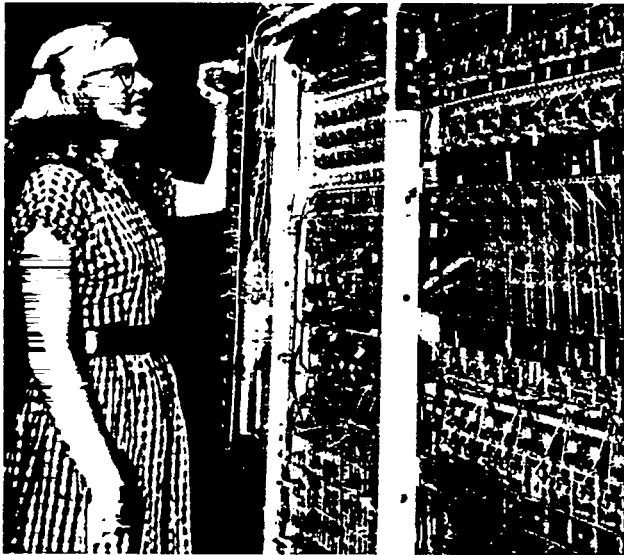


Fig. III-5.

platen-rotating Flexiwriter. This was a paper-tape input device on which the two-dimensional expressions of Madcap 3 (developed about 1960) could be typed. Perhaps the best example of all was the design of the bit manipulation instructions in the early 1960s used specifically by the set-theoretic operations of Madcap 4.

I have forgotten who (perhaps it was engineer Allan Malmberg) suggested attaching a simple amplifier to pick up noises or music (depending on your point of view) from the running computer. It was an almost indispensable diagnostic device for both MANIAC I and MANIAC II. Nick tells the story of the time that he was chatting with Bob Richtmyer in a corner of the MANIAC room while one of their programs was running. Bob, besides being a well-known theoretical physicist, is also a noted musician. As they were chatting, Bob heard a slight change in the sound emanating from the amplifier and announced that he thought box 19 was being skipped. (He was referring to flow diagram box 19.) Sure enough, upon examination they discovered a computer malfunction was preventing entry to box 19.

Let me point out another interesting feature of MANIAC. Note that the register flipflops with attached neon lights were on the front of the machine

(see Fig. III-4), and so the binary digits* that they contained could be seen directly. The whole machine was, in effect, part of its console. Furthermore, by using clip leads to short one side of the vacuum-tube flipflops, you could actually change the contents of a register (while the machine was stopped). A skilled operator, like Don Bradford or Verna Ellingson Gardiner, could fetch a word into a register, make a change in it using clip leads, and store it back in memory in about the same time that most modern operators can type a one-line command on a terminal.

Figure III-6 shows another of the early coders, Marj Jones Devaney, working at the I/O station. In



Fig. III-6.

front of her is a teletype printer and to her left is a mechanical paper-tape reader. Later, that reader was replaced by a more reliable Feranti photoelectric reader. In general, we found what is well known today that electronic equipment is preferable to mechanical equipment.

The fast line printer for MANIAC I is shown in Fig. III-7. The Analex (serial number 1) was a cylinder printer capable of putting out up to 10 lines per

*We called them "bigits" for a short time before common adoption of the term "bit."

We believe that the first "performance measurement" was done on MANIAC I. Using an interpretive approach, Gene Herbst (now with IBM), Nick, and I were able to get a dynamic count of the instructions used in various calculations. Table III-I

TABLE III-I

ANALYSIS OF THE CODE FOR A PROBLEM IN
HYDRODYNAMICS

Vocabulary Symbol	Static Count	Percentage of Static Count	Dynamic Count	Percentage of Dynamic Count	Time	Percentage of Time
AA	156	13.5	3499	12.3	314.9	5.4
AB	8	0.6	309	1.0	27.8	0.4
AC	0	0.0	0	0.0	0.0	0.0
AD	0	0.0	0	0.0	0.0	0.0
AE	0	0.0	0	0.0	0.0	0.0
AF	1	0.0	40	0.1	3.6	0.0
BA	93	8.0	2745	9.6	247.0	4.2
BB	68	5.8	2149	7.5	193.4	3.3
BC	0	0.0	0	0.0	0.0	0.0
BD	0	0.0	0	0.0	0.0	0.0
BE	0	0.0	0	0.0	0.0	0.0
BF	4	0.3	4	0.0	0.3	0.0
CA	34	2.9	332	1.1	16.6	0.2
CB	49	4.2	936	3.3	46.8	0.8
CC	13	1.1	447	1.5	20.1	0.3
CD	24	2.0	770	2.7	34.6	0.6
CE	0	0.0	0	0.0	0.0	0.0
CF	0	0.0	0	0.0	0.0	0.0
DA	93	8.0	2498	8.8	2592.9	45.1
DB	3	0.2	80	0.2	83.0	1.4
DC	157	13.6	4163	14.6	249.7	4.3
DD	49	4.2	1154	4.0	1197.8	20.8
DE	28	2.4	763	2.6	106.8	1.8
DF	6	0.5	12	0.0	0.8	0.0
EA	0	0.0	0	0.0	0.0	0.0
EB	118	10.2	3225	11.3	209.6	3.6
EC	104	9.0	2430	8.5	160.3	2.7
ED	13	1.1	37	0.1	1.4	0.0
EE	11	0.9	407	1.4	53.7	0.9
EF	19	1.6	86	0.3	6.4	0.1
FA	38	3.2	1101	3.8	88.0	1.5
FB	42	3.6	1063	3.7	85.0	1.4
FC	0	0.0	0	0.0	0.0	0.0
FD	0	0.0	0	0.0	0.0	0.0
FF	17	1.4	0	0.0	0.0	0.0
800	3	0.2	83	0.2	6.1	0.1
Totals	1151		28333		5747.4	

shows the results for a hydrodynamics calculation. The very high percentage of time (45.1%) used by the multiply instruction (DA) was noted for input to the design of MANIAC II.

Not all of the computing on MANIAC I was numerical. We also had some fun with combinatorial problems. We wrote a code for the queens problem in chess, just as most students do today, and calculated the 92 solutions for the 8 by 8 board (one solution is shown in Fig. III-9). At the time, I was too inexperienced to program on MANIAC the group operations with which to calculate the solutions inequivalent under reflections and rotations; I can remember spending an afternoon in my office with the 92 machine-produced solutions and my chessboard grinding out the 12 inequivalent solu-

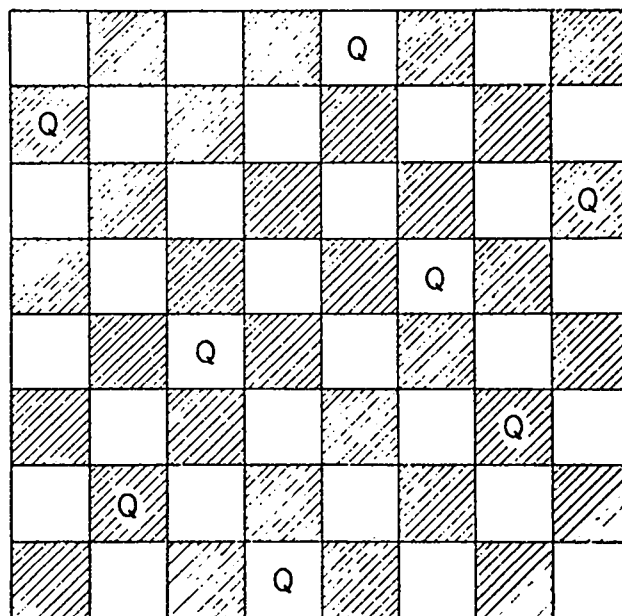


Fig. III-9.

tions by hand. I believe it was my independent discovery of backtracking in the early days of MANIAC I that nurtured the interest in combinatorial algorithms that I still have today.

We also had a chess-playing program on MANIAC I. However, because of the slow speed of MANIAC (about 10 000 instructions per second) we had to restrict play to a 6 by 6 board, removing the bishops and their pawns. Even then, moves averaged about 10 minutes for a two-move look-ahead strategy. The program played several games, both against itself and against humans; it even won one game against a beginner who had been taught how to play specifically for our experiments. We wanted to determine the level of play of the program. As I remember, we concluded that the program was equivalent to a beginner with about a half-dozen games experience. Perhaps the most exciting game was one played with Martin Kruskal, a Princeton physicist; Kruskal gave MANIAC queen odds. The game was a stand-off for some time; once after a surprising move by MANIAC, Kruskal even murmured, "I wonder why *he* did that?" In the end, however, Kruskal did win; but when he checkmated the machine at move 38, it responded with one more move, illegal of course. We were dumfounded for a while, until we traced the trouble and realized that the program had never been taught to resign. When

confronted with no moves, it got stuck in a tight loop. As some of you may recall, tight loops were often hard on electrostatic memories. In this case, the tight loop actually changed the program, creating an exit from the loop, whereupon the program found the illegal move. You might call that a "learning" program.

MANIAC I did not actually leave service until 1965,* but it was replaced at LASL in 1957 by the faster, more powerful, easier-to-use MANIAC II (Fig. III-10).

The chief advantages that the second-generation machines, MANIAC II and IBM 704, had over the

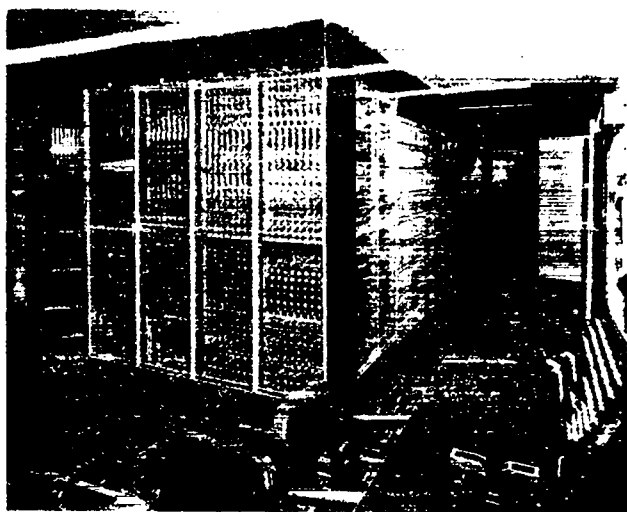


Fig. III-10.

first-generation MANIAC I and IBM 701 was floating-point arithmetic. As Ed Voohees mentioned in Sec. II, a substantial portion of the software effort on the early machines involved producing sub-routines to do our arithmetic in floating point, letting the user think more at his own level without complicated scaling. With the advent of the more powerful second-generation machines, attention could be given at a higher level, and we then saw the beginning of real programming languages; FORTRAN on the 704 (1957) and Madcap on MANIAC II (1958). However, whereas Madcap has evolved and improved over the years along with MANIAC II and with our understanding of languages and algorithms, FORTRAN has been essentially static. In 1958, Roger Lazarus and I participated in the

*It went to the University of New Mexico and was used there until 1965 when it was retired.

programming of a neutronics hydrodynamics calculation in FORTRAN to learn the language. Some of the features that annoyed us then, like required parentheses in IF statements, are still there. It is too bad FORTRAN was frozen and became a standard so early.

I am not going to say much about Madcap, because most of its development took place in the 1960s and 1970s. However, Fig. III-11 shows a small

```

Li - {24} → Li
if Li = 0:
    Si + Pi+1 Qi → Si
otherwise:
    Qi = Pi+1, nest deeper
≠ 9080      Pi Si (  $\frac{|A_i| + |K_i|}{|A_i|}$  ) → Pi
porperno ( ) = Pi ; go to exit

```

Fig. III-11.

piece of a Madcap program written in 1965. The two-dimensional features are exemplified by the subscripts and binomial coefficients. The set-theoretic notation was added in 1963 and expanded with a structure former notation in the latest Madcap in 1972. Two other features that I developed in Madcap of which I am particularly proud are type propagation in lieu of declarations and an implementation of activation record retention that allows incorporation of a very useful function data type.

I wish to conclude with a crude analogy between the floating-point libraries on the early machines and the development of FORTRAN preprocessors today. In both cases, we were or are attempting to tack on features that should have been or should be an integral part of the basic computational tool. While the hardware upgrading was accomplished fairly easily in the late 1950s, the sheer size of the computing industry today makes widespread acceptance of the conversion to higher level languages such as Algol 68, Pascal, or even Madcap, painfully slow if not impossible. I am certainly glad computers got index registers, floating-point arithmetic, and the like when they did; the added complexity of computing without these features would be considerable.

IV CONTRIBUTIONS TO MATHEMATICS

by

Roger B. Lazarus

First, I want to talk about a particular area of physics that I was involved in personally: the estimating of the energy release of nuclear devices. Then, I will just mention, for the record and for the fun of reminiscence, some of the early problems that I remember that were done in the early 1950s. Most of them, but not all, were done on the MANIAC I. Finally I want to close with some speculative remarks entitled, "Why Was It More Fun?"

The whole Los Alamos Project was started with the "estimate" that if the fission reaction released both energy and extra neutrons, then a chain reaction could be brought into existence that would give an explosion. And that is what it was all about.

In the 1940s during the war, and in the 1950s, the main challenging calculational problems were those of shock hydrodynamics and neutron transport. There was also radiation transport as a problem, and generally, that is easier than the neutron transport.

The hydrodynamics problems are described by hyperbolic nonlinear partial differential equations in space and time. It is the nature of those equations that discontinuities in the dependent variables can come about spontaneously, so that the straightforward replacing of partial differential equations with partial difference equations can run into trouble because the derivatives can become infinite.

I was not really quite sure of who did what first. I found in the preface of the 1957 edition of Richtmyer's book *Difference Methods for Initial-Value Problems*,* the following sentence. "Finite-difference methods for solving partial differential equations were discussed in 1928 in the celebrated paper of Courant, Friedrichs, and Lewy but were put to use in practical problems only about fifteen

years later under the stimulus of wartime technology and with the aid of the first automatic computers...." Well, the rest of the paragraph talks about the LASL part of the whole thing.

The accounting machines that Jack Worlton showed you, which were used primarily for shock hydrodynamics, were used only for the smooth part of the flow. The accounting machines would run as far as they could, and when it was necessary to do the shock fitting—to apply the jump conditions across the shock—that was done by hand. It was that hangup that led to the invention by Richtmyer and von Neumann of a thing called pseudo viscous pressure, which is an extra term added to the difference equations that will smear out the shock over a few zones, the number of zones being essentially independent of the shock speed and material. It will do this in such a way as to preserve the important quantities: shock strength and speed. That invention was made specifically for a calculation done on the SSEC. That was in the late 1940s. That method of smearing is still in use.

There was a two-space dimensional R-Z cylindrical geometry hydrodynamics code run on the ENIAC, and I imagine it was the first two-dimensional hydrodynamics done anywhere. But usually, in the 1940s and the 1950s, one worked with a single space variable, either spherical symmetry or the symmetry of an infinite cylinder. There are in hydrodynamic calculations both numerical instabilities and physical instabilities. Physical instabilities, such as mixing and picking up waves when one substance slides across another, are suppressed by the assumption of symmetry. This led to a very deep part of the early computing problems.

When I came to LASL in early 1951, my first assignment was to do a yield calculation on a CPC for a pure fission bomb. That machine was so simple and so incapable, in modern terms, that at least on the Model-1 CPC, it was not really possible to do

*Robert D. Richtmyer, *Difference Methods for Initial-Value Problems*, (Interscience Publishers, Division of John Wiley & Sons, 1957).

partial differential equations at all. Simplifying assumptions were made, essentially parameterizing the shape of everything, so that one could solve ordinary differential equations. But that same year, in the summer of 1951, my second job was to help with a code that had been written for SEAC for a thousand words of memory. It was really quite a substantial calculation that directly integrated the finite difference approximations to the partial differential equations. So there was quite a contrast, of which I do not remember being particularly conscious. The focus was on the application—what approximations were reasonable, what you needed to do, and then you looked around to see if you could do it.

The neutron transport part of things is described by a linear integro-differential equation in which the rate of change of the number of neutrons at a given place, direction, speed, and so on, depends on the scattering of all the neutrons at that point and going in all directions. There was quite a range of difficulty for that problem. The easiest case I can think of was to find a steady-state solution for a system of spherical symmetry with a homogeneous scatterer and within the diffusion limit, which is to say short mean free path. That really is a very simple problem. The hardest, perhaps, would be something where there was no spatial symmetry, where the scatterers were in motion, where the mean free path was long compared to the dimensions, and, as an extra difficulty, perhaps there were only a few neutrons involved so that you had a discrete function.

A problem geared to this latter class was most difficult and was what led to what is perhaps the most, or at least one of the most, far-reaching LASL inventions, namely, the Monte Carlo method. In fact, one of the most important early problems was that of initiation probability—given a slightly supercritical assembly and one neutron, what is the probability that that neutron, before being absorbed or escaping or having all its daughters escape, will lead to an explosive chain reaction? You can describe Monte Carlo easily in that transport context, which is where it is perhaps most obviously applicable, but Monte Carlo grew in conjunction with the growth of probability theory itself. Now it is extremely widespread and used far from transport problems where you are actually tracking things.

The first method that I used, on the CPC code for the neutron transport problem, was called Serber-

Wilson. I assume it was invented in part by Serber and in part by Wilson. It had a lot of hyperbolic functions, and exponential integrals that were all entangled with the hyperbolic functions; and one used those marvelous WPA (Work Project Administration) tables, which were perhaps the only good result of the Depression of the 1930s. The CPC, at least the Model-2 CPC, also had those functions. It had an electronic 604, or whatever it was, that would give you those. But the terms and expressions containing these functions were not physical expressions, and they were very difficult to scale. So when I moved from the CPC to MANIAC I, carrying over Serber-Wilson, I found myself building essentially floating-point software—automatic scaling software. It was very annoying. Luckily, this was replaced by the family of methods Bengt Carlson came up with in the early 1950s and which are still, at least generically, the primary method of choice for neutron transport. The dependent variables were the currents themselves, the neutrons per square centimeter per second for certain energies, and they were easy to scale. It was a tremendous blessing for fixed point. It also happened to be a fundamentally superior method in the long run.

From the CPC to the IBM STRETCH computer, there was approximately a factor of 10^4 increase in speed and in memory size. The factor in run time for a typical yield calculation was 10^6 . Beginning on the CPC, at the end on STRETCH, and in fact today on CRAY-1, the really difficult problems take about 15 hours and the easy ones take half an hour. Another thing that has not changed, besides the run time, is the agreement between calculation and experiment, by an appropriate measure. At least, one can observe that tests of nuclear devices still seem to be necessary. The answer to that paradox is, of course, the increase in complexity of the things we are trying to do. When we say 10% agreement, we do not mean between the same measures and quantities. It might be between some spectrum now, and it was between some single number then.

I would like to wrap up this part on estimating yields by particularly stressing the word *estimating*, the difference that is suggested by the connotation of *estimating versus calculating*. It seems to me that we were more conscious then that we were estimating something. I remember quite early thinking in terms of what I called the fudge function, which was simply a function over a parameter space

whose value is the ratio of the truth to what I calculated. I saw my job, which was to predict the energy release of some proposed design, as being a problem in interpolation or extrapolation on the fudge function. I was calculating the yield for a given design, but that is not the number I would predict. I would interpolate or extrapolate on all the parameters I could think of and get the fudge function value, 1.32 or whatever, and multiply the answer by that. The trouble, of course, is the unknown dimensionality of the parameter space, which is to say knowing which of the probably infinite number of things that were actually changed were relevant. There were certainly times when one thought that the phase of the moon was the relevant parameter that would explain the mystery.

The physics that we added as time passed and as computers grew in capability was added, in general, where this fudge function was ill-behaved or perhaps where we only had a single point. To say we had only a single point is equivalent to saying that we were starting into a new dimension, where something was to be varied for the first time. Then we could clearly not interpolate, nor could we extrapolate. We could either assume it did not matter, which in some cases was patently absurd, or we would have to add to the physics. But, all the time we were conscious of the fact that we could not add *all* the physics. If we got a correct answer, it had to imply that the errors had cancelled precisely. There was no possibility that we would calculate all the processes correctly. The process of adding physics was one usually of replacing what I always distinguished as fudge *factors* (which are usually things called the effective such and such, or the net equivalent so and so) with the relevant equation. For example, in the early days, we did not keep scattering cross sections as a function of energy. We kept the one number that was the cross section averaged over the fission spectrum or whatever approximate energy spectrum we expected.

Then, with all these fudge factors, of which we had really quite a few, there was a process of tuning the code to give an attractive fudge function, ideally, one which would be unity over all the measured points, but in any case, something that you felt comfortable with. That was part of the job: to tune the code. I remember that just before the 704 came we had a 701 code that was really nicely tuned.

When we moved to the 704, we threw away all those scale factors, all those powers of 2, with great joy because they were really hard to keep track of. Of course, the code on the 704 did not repeat, so we started debugging. We went on and on until we were really sure we had checked all the logic, and then we discovered that one of our fudge factors that we thought was in there with a 2^{+2} scale factor was in there with a 2^{-2} . Given the pressure of time, the best we could do was change the fudge factor by 16 and swallow our pride. If we had put the *correct* fudge factor into the 704 code, we would have loused up our fudge function. It stayed that way, as I remember, until the test moratorium in the late 1950s. Then we finally had time to retwiddle.

Let me now switch to just giving a list and a few comments on some of the early MANIAC I problems. MANIAC I was a 1024-word machine when it was born—two instructions per word. Later it got a drum. One of the first problems, besides the yield calculation that was my bread and butter, was something called Quintet, which was a five-dimensional integral for scattering corrections in experimental physics. I remember it took approximately a second to get the integrand. It seemed like an easy problem until you really took some nominal number of points (I think a half a dozen when we first started) and took the fifth power; then suddenly it was large. That is where I learned about Gaussian quadrature. There were, of course, no index registers then; a quintuple DO-loop involved bringing out your instruction and adding the number or subtracting the number from your address and storing it back. Well, that was minor as a problem.

A more important problem was the Fermi, Pasta, Ulam nonlinear oscillator investigation that led to what is now still quite a live field, called soliton theory. There was the first Monte Carlo equation of state that was, I think invented by Marshall and Arianna Rosenbluth (well, it is possible that Teller made the original suggestion), where you tried to calculate the actual equation of state of the material by stochastic processes on sample molecules. There was what I assume was the first stellar evolution calculation, done by Art Carson and George Gamow. There was, I think, the first chess; it was not strictly chess; it was 6 by 6 chess. It was a slightly reduced board. But it was quite important

as being, I think, the first or at least a very early class of computing that led to what is now called artificial intelligence.

There was a code worked on by Verna Gardiner with Gamow for trying to discover the code for DNA selection of amino acids, or whatever that is, which of course was not successful. Other things were virial coefficients, nuclear scattering problems, and Schrödinger equation integrations. Intranuclear cascades were another Monte Carlo thing; in a heavy nucleus one actually tried to count the production of cascades of pions within the nucleus.

I noticed when the title was mentioned, it said "mathematics." I really have thought physics was what LASL was about. But if I was really supposed to talk on math, I had better mention at least a couple of examples. There was a calculation of group characters on MANIAC I, done by Paul Stein and others. There was perhaps the beginning of what a lot of people now think of as experimental mathematics, where you try things and explore using the computer, trying to form conjectures or get some new insight. It was believed, as I suppose it is by most mathematicians today, that ideas are not worth anything until you prove something, but at least there was an important role computers could play at the experimental level. One that I was involved with, an idea of Stan Ulam's, was in the area of pseudo prime numbers: integers that have not the defining properties of prime numbers but have their same probability distribution within the set of all integers. It was discovered that many of the properties of primes apparently are due to their distribution only, not to their unique defining property. There were also some cell multiplication studies; Ulam also was involved with that.

In parallel with all these problems (probably every one I mentioned was coded in absolute), there was the business of subroutines and assembly routines. I do not really know who started what, or what credit LASL can take and get away with it, but it was just sort of common sense. You got tired of coding the sine function over and over again, so you borrowed somebody's. I remember personally being quite negative about subroutines. It seemed to me outrageous to lose control of where your numbers were in memory. That may be partly because of the 1951 SEAC experience, which was the first thing I had to do with a really modern computer. I was han-

ded a code, all written, that required 1023 words. I was told we were going to take this in August and put it on the SEAC (which actually had a 1024-word memory) and "would you please make sure that it is correct." That seemed a reasonable thing at the time. It was right there. Every bit was put down that the machine could see. So I worked for a couple of months and I found three coding errors. As it turned out, those were the only three coding errors there were. Unfortunately, I made a mistake fixing the three errors, so there was still a coding error when we got to Washington. Again unfortunately, I needed two more words to fix those three errors, and that came to 1025. Luckily that was a four-address machine—A, OP, B, C, and GO TO D—so you had a free branch on every instruction. You could put everything in where you wanted except for arrays, so I just picked a constant, decoded it, and arranged addresses so that some instruction had the numerical value and off we went.

Maybe that story leads naturally into the topic, "Why Was It More Fun?" I have six things listed here, of which the first is "Maybe it was not more fun." There is selective recollection; there is the nostalgic fallacy. It is hard to judge. When the session started, it struck me as particularly appropriate that they were doing that jackhammer work outside because it was true that that was the feeling I got of working conditions around the computer, especially at New York University with UNIVAC I in 1953, with the air full of dust. But it was fun, and what could it have been? It could be that there were no operating systems. There was more a feeling of man against the elements; that you were searching for the maximal exploitation of a set of hardware. It was just you and it. Perhaps it was that we had, I think, at least for these big problems, a deeper understanding of a simpler calculation than we have today. Today I think we have a shallower understanding of more complex calculations. Perhaps it was that there was no subdivision of labor. At least if you read some of the Marxists, they will tell you most everything is less fun now because of subdivision of labor. People do not grasp the whole of what they are doing. We certainly did then, because we did everything ourselves. Perhaps it was because there was no management or at least the management was invisible. In those days, at least according to my selective recollection, the scientist did

science, instead of management. Lastly, and perhaps more seriously, I will echo the point that I think Ed Voorhees made. There was more prior analysis and estimating of what you were doing, especially in the fixed-point computer era. There was more checking; and there was more skepticism, both

with respect to the hardware and with respect to the physics. We were so far from putting it all in. It was so new that we were not trapped into this confusion between estimating and calculating. We did not think we were getting the answer to the original physics problem.